IN THE SPECIFICATION

Presented below are specification changes showing the changes made. The changes in the specification add no new matter.

Please replace the paragraph [0049] with the following rewritten paragraph:

[0049] According to this example, the software architecture of the object application 320 includes a base agent 410, a container object 420, and a sub-component prerequisite factory 430. The base agent 410 includes logic to instantiate and execute the container object 420 that corresponds to a page associated with the base agent 410. The base agent 410 receives the page request 400. The container object outputs the sub-component output 480. In one embodiment, the base agent 410 is instantiated by the application server 310 in response to a URL request for the page associated with the base agent 410. Then, the application server 310 causes the requested page to be displayed by invoking the base agent's execute method. The base agent's execute method, in turn calls the execute method of the container object 420.

A2

Please replace the paragraph [0063] with the following rewritten paragraph:

[0063] Figure 6 is a block diagram of a software architecture for an object application according to another embodiment of the present invention. The architecture depicted seeks to facilitate the management, tracking and controlling of page prerequisites. According to this example, the software architecture of the object application 320 includes a base agent 610, a prerequisite factory 630, page objects 640, page prerequisite objects 650, a properties data store 630, and a status data store 660. Base agent 610 receives page request 600. Page objects 640 and/or page prerequisite objects 650 output the page output 670.

A3

Please replace the paragraph [0079] with the following rewritten paragraph:

[0079] Before describing various improvements to existing properties file syntax, it may be instructive to briefly discuss the current usage model of such files with referent to Figure 8. Java has a special built-in mechanism for handling internalization. Locale specific properties/attributes are placed in files having a ".properties" extension (herein referred to as a "properties file"). The properties file contains one or more lines of attribute name value pairs (hereinafter "attribute-value pairs") in the general form represented by expression 810. The first portion of the attribute-value is an attribute name, such as "prompt1", as illustrated in expression 820. The attribute name is followed by an equal sign. The final portion of the attribute value pair comprises a string value, such as "please enter your name", as illustrated in expression 820, to be associated with the particular attribute. In the Java framework, properties files may be loaded into objects of type "ResourceBundle". Then, the resulting resource bundle objects may be queried to determine a value associated with a particular attribute of interest. For example, if a properties file contained the line:

Abc.def.gh_I=j

then invoking the appropriate resource bundle object's get method, e.g., resourceBundle.getString ("*Abc.def.gh_I*") would return the string value "*j*". In this manner, applications can query the value of an attribute during runt time. However, one limitation of this prior art syntax is that both properties files and the corresponding internal resource bundle representations are flat, non-object oriented collections of key-value pairs. In connection with an object-oriented implementation of page sub-component prerequisites, page prerequisites, and/or other applications, the assignee of the present invention has found it to be advantageous to (1) allow properties/attributes to be defined and accessed on an object basis; (2) enable inheritance of properties/attributes through some hierarchy of objects; (3) specify and override

properties/attributes of objects as they are used in different contexts, and (4) enable inheritance of contexts.

Please replace the paragraph [0100] with the following rewritten paragraph:

[0100] Again, assuming the desired value was not located by a previous stage, the search enters a final stage 1430 in which context is disregarded and the object hierarchy is traversed. First the object determines if it has defined a value for *attr*, e.g., by making the call *getValue("Obj3.attr")*. If this call is unsuccessful, then a subsequent retrieval is made with reference to the object's parent, *Obj2*. If that call fails, a final attempt to retrieve the desired value will be made with reference to *Obj2*'s parent, *Obj1*, e.g., by calling *getValue("Obj1.attr")*.

Please replace the paragraph [0108] with the following rewritten paragraph:

[0108] Figure 17 is a block diagram ~~the~~ that conceptually illustrates software architectural components that enable processing of an improved properties file syntax according to one embodiment of the present invention. According to this example, an object 1710 exposes a *getValue()* method 1711 which in turn may call a *getWithContextInheritance()* method 1713 and/or a *getWithObjectInheritance()* method 1715 and/or a *getWithInheritance()* method 1717. In alternative embodiment, the *getValue()* method 1711 may be private (i.e., not exposed outside of the object 1710).

Please replace the paragraph [0109] with the following rewritten paragraph:

[0109] According to one embodiment, object 1720 inherits logic from a base object that serves as an abstraction for the interface to API 1720. In this manner, object 1710 automatically has the ability to retrieve the value of any of its attributes without needing knowledge of the

underlying interface to the properties data store, e.g., properties file 1730 (e.g., by making the call *getString 1721).* Additionally, inheritance of this logic saves design and development time.

Please replace the paragraph [0110] with the following rewritten paragraph:

[0110] In one embodiment, the functionality of the getWithContextInheritance() method 1713 and getWithObjectInheritance() method 1715 are represented by the processing described with reference to Figure 15; and the functionality of the getWithInheritance() method 1717 is represented by the processing described with reference to Figure 16.